Managing Kubernetes configurations can be complex. Policies provide a way to define standards and best practice, and ensure consistent configurations across multiple clusters. In this article, we will discuss Kyverno – a new open source policy management tool designed for Kubernetes,

# Policy Management

Kubernetes provides a declarative management interface. Users can specify the desired state of the system using API definitions, and Kubernetes controllers then work to make the current state match the desired state. This approach is used for managing cluster nodes and services as well as for managing application workloads.

A policy typically represents required configurations. Kubernetes itself provides several policies such as:

* **Network Policies**: to secure network traffic to and from pods
* **Pod Security Policy**: to control privilege and access profiles for pods
* **Quotas and Limit Range**: to manage resource utilization and fairness

These policies are native API objects that govern the behavior of the Kubernetes cluster. However, there are several situations where users want to configure settings that are required cluster-wide and even across clusters within an enterprise. These Kubernetes constructs are powerful but have no effect if not configured. And configuring them correctly, for multiple workloads is a non-trivial task.

For successful adoption and growth of enterprise Kubernetes, an organizational pattern that is emerging is that an internal platform team is formed to manage Kubernetes clusters, infrastructure for Kubernetes, and all of the common services that must run in each cluster or are shared across clusters, and are required to operationalize Kubernetes (e.g. for monitoring, logging, security, backup and DR, etc.).

The platform team provides Kubernetes services and solutions to product development teams within the enterprise. The platform team would also be responsible for setting the policies for the security and proper configuration and operations of Kubernetes clusters and workloads – for example, by mandating that resource quotas are required on all shared clusters.

To scale Kubernetes usage, it is necessary that a central team is responsible for ensuring proper configuration of Kubernetes policies, and validation of other best practice guidelines, This is why policy management is becoming increasingly important for enterprise Kubernetes.

# Being Kubernetes Native

In this section we will discuss what being Kubernetes native means, and why that important for an enterprise policy management solution.

Kubernetes is designed to be extended using [Custom Resources](https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/). Custom Resources allow us to introduce a new object type, and use it via any tool that can interact with the Kubernetes API. For example, custom resources can be specified in YAML. This allows us to apply declarative management principles using tools such as [Kustomize](https://github.com/kubernetes-sigs/kustomize).

The Custom Resource mechanism allows us to extend Kubernetes and utilize common tooling for managing Kubernetes configurations. Policy management is a form of configuration management, where a policy provides defaults for configuration and also determines what is allowed or disallowed. Hence it is important that a Kubernetes policy management solution also adopt the same practices and approach as Kubernetes management.

Existing solutions for Kubernetes policy management were initially designed for other use cases and then fitted into Kubernetes. Our work with customers has led us to learn that policy management is important to Kubernetes’ success within enterprises, and our customers wanted a simpler and more Kubernetes friendly way of defining and managing policies.

# What Kyverno does

Kyverno allows cluster administrators to validate, mutate, and generate configurations. Kyverno policies are Kubernetes resources. The policies match other resource based on Kubernetes label selectors with support for wildcards. Kyverno uses Kubernetes events to report policy enforcement.

Kyverno also monitors changes on policies, and scans existing resources for policy violations. Policy violations are reported as sub-resources on the policy object.

## Validate configurations

Here is an example of a Kyverno policy that validates that images are only pulled from an allowed list of image registries (based on wildcard patterns):

apiVersion : kyverno.io/v1alpha1

kind: Policy

metadata:

name: check-registries

spec:

rules:

- name: check-registries

resource:

kinds:

- Deployment

- StatefulSet

validate:

message: "Registry is not allowed"

pattern:

spec:

template:

spec:

containers:

- name: "\*"

# Check allowed registries

image: "\*/nirmata/\* | https://private.registry.io/\*"

## Mutate Configurations

Kyverno supports two different ways to mutate configurations. The first approach is to use a [JSON Patch](http://jsonpatch.com/) which provides a precise way to make changes to a JSON (or YAML) document.

Here is an example of a policy that adds a label to a deployment:

apiVersion : kyverno.io/v1alpha1

kind : Policy

metadata :

name : policy-deployment

spec :

rules:

- name: patch-add-label

resource:

kinds :

- Deployment

mutate:

patches:

- path: /metadata/labels/isMutated

op: add

value: "true"

The other way to mutate resources is to use an overlay with conditionals that describes the desired state. Here is a policy that checks if an image tag ends with “latest” and changes the imagePullPolicy to Always:

apiVersion: kyverno.io/v1alpha1

kind: Policy

metadata:

name: set-image-pull-policy

spec:

rules:

- name: set-image-pull-policy

resource:

kinds:

- Deployment

mutate:

overlay:

spec:

template:

spec:

containers:

# if the image tag is latest, set the imagePullPolicy to Always

- (image): "\*:latest"

imagePullPolicy: "Always"

## Generate Configurations

The final type of policy rule generates new configurations, when a new [Kubernetes namespace](https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/) is created in the cluster. This is useful to set defaults for the namespace, such as a default Kubernetes NetworkPolicy that is generated for all namespaces by the policy below:

apiVersion: kyverno.io/v1alpha1

kind: Policy

metadata:

name: "default"

spec:

rules:

- name: "deny-all-traffic"

resource:

kinds:

- Namespace

name: "\*"

generate:

kind: NetworkPolicy

name: deny-all-traffic

data:

spec:

podSelector:

matchLabels: {}

matchExpressions: []

policyTypes: []

metadata:

annotations: {}

labels:

policyname: "default"

# How Kyverno works

Kyverno installs itself as an admission controller, which receives webhook events for all API object changes. Kubernetes supports two special [dynamic admission controllers](https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/) which are designed to allow extensibility via tools like Kyverno, the MutatingAdmissionWebhook and the ValidatingAdmissionWebhook. As the names imply, controllers of these types can change and validate API objects.

# Learn More or Try Kyverno

The [Kyverno GitHub repository](https://github.com/nirmata/kyverno/) is the best place to collaborate, ask questions, and get detailed information on the project:

<https://github.com/nirmata/kyverno>

Kyverno is easy to install and try in your Kubernetes cluster! Here is a command that installs Kyverno:

kubectl create -f https://github.com/nirmata/kyverno/raw/master/definitions/install.yaml

You can then try the sample policies provided in the examples folder:

<https://github.com/nirmata/kyverno/blob/master/samples/README.md>

Complete details on how Kyverno policies work are available in the documentation:

<https://github.com/nirmata/kyverno/tree/master/documentation>